

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-168733

(43)Date of publication of application : 22.06.2001

(51)Int.Cl.

H03M 13/09
G06F 11/10

(21)Application number : 2000-312558

(71)Applicant : THOMSON CSF

(22)Date of filing : 12.10.2000

(72)Inventor : LAURENT PIERRE-ANDRE

(30)Priority

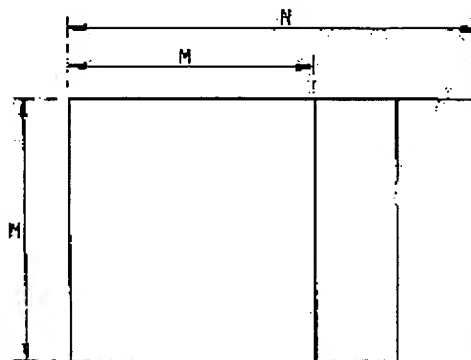
Priority number : 1999 9912710 Priority date : 12.10.1999 Priority country : FR

(54) PROCESS FOR CONSTRUCTING AND CODING LDPC CODE

(57)Abstract:

PROBLEM TO BE SOLVED: To easily construct an LDPC code for protecting a binary information string.

SOLUTION: Respective information strings are composed of the N pieces of symbols decomposed into the N-M pieces of useful information symbols X_i and the M pieces of redundant information symbols Y_m and respective codes are defined by an inspection matrix A composed of N columns and $M=N-K$ rows provided with the t pieces of non-zero symbols inside the respective columns. In this process, the same number of the non-zero symbols are allocated to all the rows of the inspection matrix A, the number t of the symbols is an odd number as small as possible, the column is defined by a method that the optional two columns of the inspection matrix A are provided with only one non-zero value at maximum and the row is defined by the method that the two rows of the inspection matrix A are provided with only one non-zero common value.



(19)日本国特許庁 (J P)

(12) 公 開 特 許 公 報 (A)

(11)特許出願公開番号

特開2001-168733

(P2001-168733A)

(43)公開日 平成13年6月22日(2001.6.22)

(51)Int.Cl.⁷

識別記号

F I

デマコト*(参考)

H 0 3 M 13/09

H 0 3 M 13/09

G 0 6 F 11/10

3 3 0

G 0 6 F 11/10

3 3 0 S

審査請求 未請求 請求項の数4 O L (全 17 頁)

(21)出願番号 特願2000-312558(P2000-312558)

(22)出願日 平成12年10月12日(2000.10.12)

(31)優先権主張番号 9 9 1 2 7 1 0

(32)優先日 平成11年10月12日(1999.10.12)

(33)優先権主張国 フランス (F R)

(71)出願人 591000827

トムソン-セーエスエフ

THOMSON-CSF

フランス国、75008・パリ、ブルバール・

オースマン・173

(72)発明者 ビエール アンドレ ローラン

フランス国 95550 ベッサンクール、

シュマン デ ムニエ 114

(74)代理人 100109726

弁理士 園田 吉隆 (外1名)

(54)【発明の名称】 LDPCコードの構築およびコーディングのためのプロセス

(57)【要約】

【課題】 バイナリ情報列を保護するためのLDPCコードを簡易に構築する。

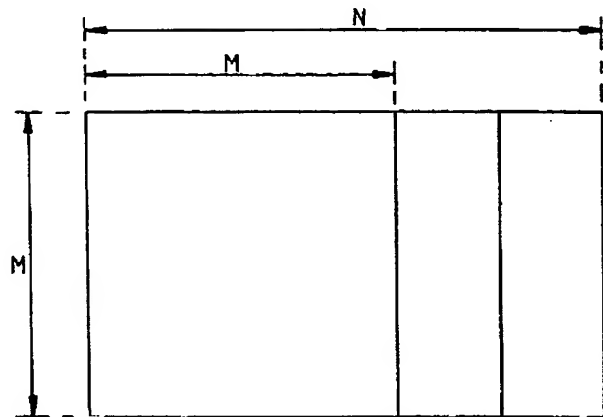
【解決手段】 各情報列が $N-M$ 個の有用な情報シンボル X_i と M 個の冗長情報シンボル Y_m とに分解される N 個のシンボルからなり、各コードが、 N 列からなり、各列内にも個の非ゼロシンボルを有する $M=N-K$ 行からなる検査マトリクス A により定義され、

— 検査マトリクス A の全ての行に、同じ数の非ゼロシンボルを割り当て、

— シンボルの数 m を可能な限り小さい奇数にとり、

— 検査マトリクス A の任意の2列が、多くても1つのみの非ゼロ値を有するような方法で、列を定義し、

— 検査マトリクス A の2つの行が1つのみの非ゼロ共通値を有するような方法で、行を定義するプロセスを提案する。



【特許請求の範囲】

【請求項1】 バイナリ情報列を保護するためのLDPCコードを構築するプロセスであって、前記各情報列がN-M個の有用な情報シンボル X_i とM個の冗長情報シンボル Y_m とに分解されるN個のシンボルからなり、各コードが、N列からなり、各列内にも個の非ゼロシンボルを有する $M=N-K$ 行からなる検査マトリクスAにより定義され、

- 検査マトリクスAの全ての行に、同じ数の非ゼロシンボルを割り当て、
- シンボルの数 t を可能な限り小さい奇数にとり、
- 検査マトリクスAの任意の2列が、多くても1つのみの非ゼロ値を有するような方法で、列を定義し、
- 検査マトリクスAの2つの行が1つのみの非ゼロ共通値を有するような方法で、行を定義することを特徴とするプロセス。

【請求項2】 — P行P列の n 個のサブマトリクスを形成するために、M行N列の検査マトリクスAを、M行P列の n 個のサブマトリクスに再分割し、 m^2 個の左側サブマトリクスを $M \times M$ サブマトリクスに、他の部分を $n-m$ 個の $M \times P$ サブマトリクスにグループ化し、

— 自己相関と列ベクトル w の周期的な相互相関を行うことにより、 t 個の非ゼロ値と、 $(M-t)$ 個のゼロ値とを具備する長さMの列ベクトル $w[0 \dots n-1]$ の $M \times n$ 個の数列を定義することを特徴とする請求項1記載のプロセス。

【請求項3】 — P行P列の $n \times m$ 個のサブマトリクスを形成するために、M行N列の検査マトリクスAをM行P列の n 個のサブマトリクスに再分割し、 m^2 個の左側サブマトリクスを $M \times M$ 個のサブマトリクスに、他の部分を $n-m$ 個の $M \times P$ サブマトリクスにグループ化し、

- t 個の非ゼロ値と $(M-t)$ 個のゼロ値とを有する長さMの列ベクトル $w[0 \dots n-m]$ の $n-m+1$ 個の数列を定義し、
- シフトされた数列 $w[0]$ が0または1点における場合を除いてシフトされていない数列自体と一致しないような方法で、第1の数列 $w[0]$ が0、1、または、値 t に等しい周期的な自己相関によって得られ、
- $n-m$ 個の以下のシーケンス $w[i][k]$ が、
- m の倍数によってシフトされた数列 $w[i]$ の値が、そのシフトされていない数列自体と決して一致しないような方法で、ゼロ値または値 t に等しい周期的な自己相関によって、
- かつ、 m の倍数によってシフトされまたはシフトされていない数列 i が、0または1点における場合を除いて数列 j と一致しないような方法で、 m によって多数回シフトするための数列 $w[1 \dots n-m]$ を用いて、ゼロまたは1の値の周期的な相互相関によって得られることを特徴とする請求項1記載のプロセス。

【請求項4】 有用な情報 X_i のコーディングのために、検査マトリクスAに、N-M個の情報シンボル X_i を表す列ベクトルをかけた積に等しいスイッチングマトリクス Z_m を決定し、該情報シンボルに、スイッチングマトリクス Z_m と検査マトリクスの寸法 $M \times M$ の部分の反転に等しいマトリクスBとをかけた積の結果として得られる冗長シンボル Y_m を追加することを特徴とする請求項1から請求項4のいずれかに記載のプロセス。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】この発明は、「低密度パリティ検査」を表す略語LDPCとして知られるコードの構築およびコーディングのための、簡易かつシステムティックなプロセスに関するものである。

【0002】

【従来の技術】1963年頃に提案されたギャラガーのコードは、現在、ターボコード(turbo code)に代わるものと考えられるLDPCコードの元祖である。M.J.C. MacKayによる"Good Error Correcting Codes Based on very Sparse Matrices"と題された、IEEE journal Transaction on Information Theory, Vol.45 No.2, March 1999に公開された記事は、これらのコードに関して興味深い結論、特に、

- 大きなサイズのブロックに対して、それらは、漸近的に「非常に良好なコード」であり、
- 重み付デコーディング（または、「ソフトデコーディング」あるいは「フレキシブルデコーディング」）を実行することが容易であるという事実を示している。しかしながら、それらを構築するためには帰納的な方法以外には方法は存在しない。

【0003】このコーディング技術によれば、コード (N, K) は、K個の自由なシンボルを含むN個のシンボルが、 $M=N-K$ 行、N列からなる、そのパリティ検査マトリクスAによって定義される。

【0004】この検査マトリクスAは、その低い「密度」を特徴とし、このことは、少数の非ゼロ要素を有することを意味する。さらに詳細には、このマトリクスAは、ちょうど t 個の非ゼロシンボルを各列内に有し、他の全ての要素はゼロである。

【0005】コードワードのシンボルが、 c_i 、 $i=0 \dots N-1$ と表示され、検査マトリクスの要素が A_{ij} と表示されるならば、コードは、

【数1】

$$\sum_{i=0 \dots N-1} A_{mi} c_i \quad \text{ここで } m=0 \dots M-1$$

の形態の $M=N-K$ 個の関係を満足する。

【0006】

【発明が解決しようとする課題】M.J.C. MacKayにより提案された方法は、より小さい単位マトリクスまたは三重対角マトリクスから初期マトリクスAを構築し、その

後、所望の結果に到達するように、それらの列を入れ替えるものである。しかしながら、経験的には、それらの構造に対して印加される種々の拘束条件を満足することは困難であることが示されている。この発明の目的は、上述した欠点を緩和することである。

【0007】

【課題を解決するための手段】この目的を達成するために、この発明は、K個の自由なシンボルを含むN個のシンボルを具備するLDPCコードを構築するためのプロセスであって、各コードをM=N-K行、N列の、各行にt個の非ゼロシンボルを有する検査マトリクスAにより決定し、

- a — 検査マトリクスAの全ての列に、同じ数の非ゼロシンボル「t」を割り当て、
- b — シンボル「t」の数として、可能な限り小さい奇数を採用し、
- c — 検査マトリクスAの任意の2つの列が、多くても1つの非ゼロ値を有するように、各列を定義し、
- d — 検査マトリクスの2つの行が1つのみの非ゼロ共通値を有するように、各行を定義することを特徴としている。

【0008】この発明に係るプロセスは、可能な限り低密度を有すると同時に、その必要な計算電力が数tに比例する妥当な複雑さに対して、良好な性能を与える検査マトリクスAを使用することにより、コーディングおよびデコーディングアルゴリズムを簡易化することができるという利点を有する。いくつかの誤差が存在するという限りにおいて、上記拘束条件“c”は、全ての状況において、デコーディングアルゴリズムを収束させることができる。

【0009】

【発明の実施の形態】この発明の他の特徴および利点は、添付図面に関連して以下に示された説明により明らかになる。図1は、検査マトリクスAの分割を示す配列である。図2および図3は、図1の配列の m^2 個の左側のサブマトリクスの、 $M \times M$ サブマトリクス、および、 $n-m$ 個の $M \times P$ サブマトリクスへのグループ化を示し

$$\sum_{k=0 \dots M-1} w[i][k] w[i][k] = t \quad (\text{定義により})$$

$$\sum_{k=0 \dots M-1} w[i][k] w[i][(k+p) \bmod M] = 0 \text{ 又は } 1, \text{ ここで } p = 1 \dots M-1$$

となるような、0、1またはtに等しい周期的な自己相関（シフトされた数列iは、0または1点における場合を除き、シフトされていないその数列i自体と一致しない）、

$$\sum_{k=0 \dots M-1} w[i][k] w[j][(k+p) \bmod M] = 0 \text{ 又は } 1, \text{ ここで } p = 0 \dots M-1$$

となるような、0または1に等しい周期的な相互相関（シフトされた数列iは、0または1点における場合を除き、数列jと一致しない）により得られる。

【0015】数列wを計算するアルゴリズムは非常に簡

易である。図4および図5は、この発明に係るプロセスの第1および第2の変形例に従って、それぞれ得られた検査マトリクスAを示している。図6～図9は冗長シンボルを計算するスイッチングマトリクスを示している。

【0010】この発明に係るプロセスを実施するために、検査マトリクスAが、図1に示されるように、 $N = nP$ 、 $M = mP$ であり、nおよびmが、互いに素の数となるように、n個のM行P列のサブマトリクス、または、 $m \times n$ 個のP行P列の正方サブマトリクスに再分割される。

【0011】 m^2 個の左側サブマトリクスは、その後、図2に示されるように、 $M \times M$ サブマトリクス（このサブマトリクスは、コーディングアルゴリズムを非常に簡単にすることができる。）にまとめられ、他のサブマトリクスは $n-m$ 個の $M \times P$ サブマトリクスにグループ化される。この構築プロセスは、 $m=1$ または $m=t$ による2つの変形例に従って、以下に説明される。

【0012】異なる値のmは、trが整数であることを要求する条件“a”のために、ここでは考慮しない。すなわち、 $tr = tN/M$ または $tr = tn/m$ である。nおよびmは、互いに素であり、tはmで割り切れなければならないが、したがって、mは、素数でありかつ小さいtに対しては、1またはtに等しい場合のみが可能である（小さい値のt、すなわち、3、5、7に対して真。）。

【0013】 $m=1$ （冗長性 $r/(r-1)$ を有するコード）の第1の変形例において、この発明に係るプロセスは、冗長シンボルの数が正確に N/r （rは整数）個である $r/(r-1)$ の形態の冗長度 N/K を有するコードに対して当てはまる。この場合には、MはPに等しく、図2の配列は、図3の配列に減じられる。したがって、プロセスは、t個の1と（M-t）個の0とからなる長さMのn個の数列を検索することになる。

【0014】したがって、以下、 $w[0 \dots n-1]$ と表されるこれらの数列は、

- 全ての $i=0 \dots n-1$ に対して、
- 【数2】

- $i=0 \dots n-1$ 、 $j=0 \dots n-1$ が異なる全ての対 $\{i, j\}$ に対して、

【数3】

易である。このアルゴリズムは $pos[x][0] = 0$ 、 $pos[x][1] = 1, \dots, pos[x][t-1] = t-1$ から始まって、自己相関および相互相関条件を満たすように、それらを修正することによ

り、これらの数列が「1」を有する位置 $pos[0]$ $[0 \dots t-1]$, $pos[1]$ $[0 \dots t-1]$, ..., $pos[n-1]$ $[0 \dots t-1]$ を連続して決定する。 $t=3$ に対して、使用される計算ループは、付録1に示されている。

【0016】このアルゴリズムは、 n が大きすぎると失敗する。所定の M に対して、いくつかの「小さい」コードが発見されるが、一般には大きなサイズ ($N \gg 100$) のコードが求められるので、このことはあまり重要ではない。

【0017】したがって、マトリクス A の列は、非常に簡単に、周期的に入れ替えられたベクトル w であり、
 — k 番目のサブマトリクス ($k=0 \dots n-1$) は、

$A[\text{行}][\text{列}] = w[k][(\text{行} - (\text{列} - kP)) \bmod M]$

ここで、 $\text{行} = 0 \dots M-1$ 、 $\text{列} = kP \dots (k+1)P-1$ である。

【0018】したがって、マトリクス A の各行は、正確にも個の非ゼロ値を、 n 個のサブマトリクスの各々に有し、すなわち、 n の総数 $= tr$ である。

【0019】LDPCコード ($75, 50, t=3, tr=9$)、冗長度 $3/2$ ($r=n=3$)、 $P=25$ に対

$$\sum_{k=0 \dots M-1} w[0][k]w[0][k] = t \quad (\text{定義により})$$

$$\sum_{k=0 \dots M-1} w[0][k]w[0][(k+p) \bmod M] = 0 \quad \text{又は} \quad 1,$$

ここで、 $p=1 \dots M-1$

のように、(シフトされた数列 0 が、 0 または 1 点における場合を除き、そのシフトされていない数列自体と一致しない) $0, 1$ または t に等しい周期的な自己相関によって得られる。

$$\sum_{k=0 \dots M-1} w[i][k]w[i][k] = t \quad (\text{定義により})$$

$$\sum_{k=0 \dots M-1} w[i][k]w[i][(k+pm) \bmod M] = 0, \text{ ここで } p=1 \dots P-1$$

のように、 m の倍数のシフト (m の倍数によりシフトされた数列 i は、シフトされていないもの自体と決して一致しない。) に対して 0 または t に等しい周期的な自己

$$\sum_{k=0 \dots M-1} w[i][k]w[0][(k+p) \bmod M] = 0 \quad \text{又は} \quad 1, \text{ ここで } p=0 \dots M-1$$

のように、数列 $w[0]$ (シフトされまたはシフトされていない数列 i は、 0 または 1 点における場合を除き、数列 0 と一致しない。) を用いた、点 0 または 1 に等しい周期的な相互相関、

$$\sum_{k=0 \dots M-1} w[i][k]w[j][(k+pm) \bmod M] = 0 \quad \text{又は} \quad 1, \text{ ここで,}$$

$$p=0 \dots P-1$$

のように、 m の倍数のシフト (m の倍数によりシフトされまたはシフトされていない数列 i は、点 0 または 1 における場合を除き、数列 j と一致しない。) に対して、

して、このプロセスにより得られる、例としてのマトリクス A が、図4に示されている。ここに示された配列によれば、

$w[0][i]=1$ ここで $i=0, 1, 3$
 $w[1][i]=1$ ここで $i=0, 4, 9$
 $w[2][i]=1$ ここで $i=0, 6, 13$
 である。

【0020】提案された構造は、

- 各列が、正確にも個の非ゼロ値 (ベクトル w の定義による) を有し、
- 各行が、正確にも r 個の非ゼロ値 (ベクトル w の自己相関および相互相関特性による) を有し、
- 全ての別々の列の対が、多くても1つの共通の非ゼロ値 (同上) を有し、
- 全ての別々の行の対が、多くても1つの共通の非ゼロ値 (同上) を有することを保証する。

【0021】 $m=t$ の場合に対応する上記変形例により示唆された第2の変形例によれば、この発明に係るプロセスは、 $w[0 \dots n-m]$ と表される、 t 個の「1」と $(M-t)$ 個の「0」とからなる長さ M の $n-m+1$ 個の数列を検索する。

【0022】第1の数列 $w[0]$ は、
 【数4】

【0023】実際には、 $m=1$ に対するものと同じ定義である。以下の数列 $w[1 \dots n-m]$ は、

- 全ての $i=1 \dots n-m$ に対して、
 【数5】

相関、

- 全ての $i=1 \dots n-m$ に対して、
 【数6】

- および、 $i=1 \dots n-m$ と $j=1 \dots n-m$ とが異なる全ての対 $\{i, j\}$ に対して、
 【数7】

0 または 1 に等しい数列 $w[1 \dots n-m]$ を用いた周期的な相互相関によって得られる。

【0024】数列 w を計算するためのアルゴリズムは、

上述したものと同一である。自己相関および相互相関基準のみが変更され、Mの代わりに、P個の点においてのみ、それらを確認することが必要である。

【0025】したがって、マトリクスAの列は、1またはmに等しいステップを用いて、

— AのM×M個の左側サブマトリクス：

A[行][列] = w[0][(行-列) modulo M]

ここで、行=0...M-1

列=0...M-1

— k=m...n-1に対して、

A[行][列] = w[k-m+1][行-m(列-k P) modulo M]

ここで、行=0...M-1

列=k P... (k+1) P-1

となるように、周期的に入れ替えられるベクトルwである。

【0026】このように、マトリクスAの各行は、その最初のM個の列に、正確にm=t個の非ゼロ値、したがって、P個の連続した列のn-m個の packets 毎に1個の非ゼロ値を有し、すなわち、nの総数またはtrである。

$$\sum_{i=0 \dots M-1} A_{mi} Y_i + \sum_{i=M \dots N-1} A_{mi} X_i = 0, \text{ここで } m=0 \dots M-1$$

又は、

$$\sum_{i=0 \dots M-1} A_{mi} Y_i = Z_m, \text{ここで } m=0 \dots M-1$$

又、ここで、

$$Z_m = - \sum_{i=M \dots N-1} A_{mi} X_i, \text{ここで } m=0 \dots M-1$$

【0030】したがって、プロセスは、最初に、スイッチングマトリクスのM個の量Z_mを計算し、その後、冗長シンボル：

【数9】

$$Y_m = \sum_{i=0 \dots M-1} B_{mi} Z_i, \text{ここで } m=0 \dots M-1$$

を計算する。

【0031】例えば、(75, 50)というLDPCコードに対して、量Z_mは、図6の配列により定義された等式システムを用いて計算され、該図6の配列は、その解法後に、図7の冗長シンボルの配列に変換される。

【0032】一般的な要素B_{ij}を有するマトリクスBは、マトリクスAの(寸法M×Mの)左側部分A_Mを反

$$\sum_{i=0 \dots M-1} w[0][i] b[(i+k) \text{ modulo } M] = 1 \text{ (k=0の場合)}$$

$$, 0 \text{ (k=1...M-1の場合)}$$

【0034】例えば、(75, 50)というLDPCコードに対して、冗長係数Y_mは、解法後に図9の配列に

【0027】この発明に係るプロセスの第2の変形例により得られた例示したマトリクスAは、LDPCコード(75, 30, t=3, tr=5)、冗長度5/2(n=5, m=3)、P=15に対して、図5に示される。

図5の配列において、

w[0][i]=1 ここで i=0, 1, 3

w[1][i]=1 ここで i=0, 4, 8

w[2][i]=1 ここで i=0, 5, 10

であることを明記しておく。

【0028】2つの変形例に基づいて示されたこの発明に係るプロセスは、冗長シンボルY_iおよび情報シンボルX_iをコーディングするための非常に簡易なアルゴリズムに直接つながるものである。このためには、冗長シンボルY_iをコードワードの最初のM個のシンボルであると考え、自由シンボルX_i(情報)を最後のN-M個のシンボルであると考えただけで十分である。

【0029】したがって、全てのコードワードによって満足されなければならない等式は、以下の形態に書き直すことができる。

【数8】

転したものである。該マトリクスAは、構築により、非常に簡易な形態を有し、その全ての列は、数列w[0][0...M-1]の周期的な繰返しである。

A_{ij} = w[0][(i-j) modulo M], i=0...M-1, j=0...M-1

【0033】したがって、マトリクスBは、単一の行b[0...M-1]の周期的な入れ替えであるM個の行から構成されている。すなわち、

B_{ij} = b[(j-i) modulo M]

Bは、A_Mの反転であり、係数bは以下のように定義される。

【数10】

変換される図8の配列により定義される式のシステムを介して計算される。

【0035】しかしながら、計算が不可能な場合がある。以下の形態で定義される等式を記述することは、実際には不可能である。

$$t A_M t \{b[0], b[1], \dots, b[M-1]\} = t \{1, 0, 0, \dots, 0\}$$

【0036】マトリクス $t A_M$ は、循環マトリクスであり、その第1行は、 $a[0 \dots M-1] = w[0]$ に等しい。その行列式は、そのM個の固有値 $1, 0, \dots, M-1$ の積に等しい。

【0037】k番目の固有値は、それ自体、以下の式で与えられる。

【数11】

$$\lambda_k = \sum_{i=0, \dots, M-1} a[i] \alpha_{ik}$$

ここで、 α は、1のM乗根である。

【0038】例えば、

— $w[0] = a[0 \dots M-1] = \{1, 1, 0, 1, 0, 0, 0, \dots\}$ に対し、
— バイナリコード（これらは、加算が排他的OR（XOR）に等しく、積算が論理ANDに等しいガロア体CG（2）に存在する）に対して、 $\lambda_1 = 1 + \alpha + \alpha^3$ を得る。

【0039】Mが7の倍数である場合には、等式 $1 + \alpha + \alpha^3 = 0$ は、 α が1の7乗根であるガロア体（多項式 $g(x) = 1 + x + x^3$ は単純化できず、かつ、CG（2）において原始的なものであり、ガロア体CG（23）を生じる。）を定義し、 $\lambda_1 = 0$ であることを意味している。

【0040】したがって、提案されたアルゴリズムにより発見されたLDPCコードの内、

— $t A_M$ の固有値の1つがゼロとなり、
— したがって、その行列式がゼロとなり、
— したがって、適当な値 $b[i]$ を発見することができず、
— したがって、 $(Y_i)_s$ を計算するための）コーディングを行うことができないので、この $w[0]$ を保持する場合には、Mが7の倍数であるものを消去する必要がある。

【0041】ごく一般的には、 $w[0]$ に対してなされる選択にかかわらず、コーディングが実行されることを許容しないために、適当ではないMの値が存在することになる。（ $x^M - 1$ および

【数12】

$$a(x) = \sum_{i=0, \dots, M-1} a[i] x^i$$

を因数分解することにより、）これらのMの値が、 $a(x)$ で $x^M - 1$ が割り切れる値M0の倍数であることが容易に示される。

【0042】例えば、 $t=3$ によるバイナリコードに対して、

$$-w[0] = \{1, 1, 0, 1, \dots\}$$

$$-w[0] = \{1, 0, 1, 1, \dots\}$$

は、Mが7の倍数であることを禁止する（ $a(x)$ は1の7乗根（7th root of unity）を定義する）。

$$-w[0] = \{1, 1, 0, 0, 1, \dots\}$$

$$-w[0] = \{1, 0, 0, 1, 1, \dots\}$$

は、Mが15の倍数であることを禁止する（ $a(x)$ は1の15乗根を定義する）。

— $w[0] = \{1, 0, 1, 0, 1, \dots\}$ は受け入れられない（正しくない自己相関）。

$$-w[0] = \{1, 1, 0, 0, 0, 1, \dots\}$$

$$-w[0] = \{1, 0, 0, 0, 1, 1, \dots\}$$

は、Mが3の倍数であることを禁止する（ $a(x)$ は $1 + x + x^2$ の倍数であり、1の立方根を定義する）。

$$-w[0] = \{1, 0, 1, 0, 0, 1, \dots\}$$

$$-w[0] = \{1, 0, 0, 1, 0, 1, \dots\}$$

は、Mが31の倍数であることを禁止する（ $a(x)$ は1の31乗根を定義する）。

【0043】係数 $b[i]$ は、以下のようにして計算される。Mの禁止されていない値に対して、 $a[i]$ （または $w[0][0 \dots M-1]$ ）から $b[i]$ を計算するための特に簡易なアルゴリズムが存在する。このアルゴリズムは、係数 $a[M-1, M-2, \dots, 1, 0]$ を有する有限時間インパルス応答フィルタ（FIR） $A(z)$ により、時間分割されかつフィルタリングされた後に、一連の $b[i]$ が時分割数列 $\{1, 0, 0, \dots\}$ を与えなければならないという観測によるものである。実際には、前に枚挙された $w[0]$ の内の1つを使用したバイナリコードに対して、この数列が、長さ7（または15または31または63）の最大数列（最大長数列）の連鎖によって形成される。

【0044】したがって、無限インパルス応答フィルタ（IIR） $1/A(z)$ のインパルス応答が計算され、一旦時分割されると、 $A(z)$ によるフィルタリング後に数列 $\{1, 0, 0, \dots\}$ を与える長さMのスライスがそこから抽出される。

【0045】例えば、 $t=3$ のバイナリコードに対して、かつ、 $a[0]$ 、 $a[k1]$ および $a[k2]$ のみがゼロではないものに対して、相当するアルゴリズムが、付録2に提供されている。

【0046】各コーディング時に前の計算を行わないように簡易化する目的から、コーディングアルゴリズムは、循環（recurrence： $1/A(z)$ によるフィルタリング）によって、全ての他のものを再計算可能とする、最後の $k2$ 個の要素 $b[M-k2 \dots M-1]$ により定義されさえることができる。

【0047】標準コーディングの第2段階と同様に、アルゴリズム（ZからのYの計算）は、平均して、大きなコードに対して相当な回数となる $M^2/2$ 回の操作を具備し、その複雑さは、サイズの二次関数となり、さら

に、中間配列 Z (M 個の要素)を格納し、配列 b (これも M 個の要素)を知ることが必要であるので、その場で計算されない場合には、アルゴリズムのこの部分は、

(例えば、(75, 50) LDPCコードに対する) 図9の配列により示された方法で、 Y を与える等式を再記述することによって、非常に小さいサイズの2つの中間配列のみを使用するように修正されてもよい。

【0048】($t=3$ に対して) 最初の $M-k2$ 行は、解法前に Y を与える等式のシステムの最後の $M-k2$ 行である。最後の $k2$ 行は、解法後に Y を与える等式のシステムの最後の $k2$ 行である。したがって、 Y を逆順に、すなわち、 $Y[M-1]$, $Y[M-2]$, ..., $Y[0]$ の順に計算するだけで十分である。

【0049】したがって、行われるべき操作の回数は、平均して、 $k2M/2$ 回 ($Y[M-1]$, ..., $Y[M-k2]$ の計算)とその後の $t(M-k2)$ 回(他の全ての計算)、すなわち、約 $(t+k2/2)M$ 回であり、これにより、その複雑さは、サイズの線形関数のみとなる。アルゴリズムは、入力として $X[M...n]$ を使用する。

【0050】 X の下部($X[0...M-1]$)は、 Z のための一時的な格納場所として使用され、 X

$[0...M-1]$ は、最終段階における循環シフトを回避するように、 $Z[k2...M-1, 0...k2-1]$ を格納する。 $b[i]$ は、その場で、 $b[M-k2...M-1]$ から繰返して計算される。

【0051】コードは、以下の2つの配列により定義される。

— b の最後の $k2$ 個の要素からなる配列 $endB$

$[0...k2-1]$ と、

— 数列 $w[0], w[1], \dots, w[n-m]$ の非ゼロ要素の位置を含む配列 $pos[0...(n-m+1)t]$ である。

【0052】以下の2つのサイズ $k2$ からなる内部バッ

$R=4/1$	すなわち、	4. 000	(105コード)
$R=5/2$	すなわち、	2. 500	(82 コード)
$R=6/3$ または $2/1$	すなわち、	2. 000	(203コード)
$R=7/4$	すなわち、	1. 750	(55 コード)
$R=8/5$	すなわち、	1. 600	(47 コード)
$R=9/6$ または $3/2$	すなわち、	1. 500	(124コード)
$R=10/7$	すなわち、	1. 428	(34 コード)
$R=11/8$	すなわち、	1. 375	(28 コード)
$R=12/9$ または $4/3$	すなわち、	1. 333	(84 コード)
$R=13/10$	すなわち、	1. 300	(20 コード)
$R=14/11$	すなわち、	1. 273	(17 コード)
$R=15/12$ または $5/4$	すなわち、	1. 250	(56 コード)
$R=16/13$	すなわち、	1. 231	(11 コード)
$R=17/14$	すなわち、	1. 214	(7 コード)
$R=18/15$ または $6/5$	すなわち、	1. 200	(34 コード)
$R=19/16$	すなわち、	1. 187	(3 コード)

ファが使用される。

— $b[i]$ を計算するための $reg[0...k2-1]$ と、

— $Y[M-k2...M-1]$ の中間値を格納するための $temp[0...k2-1]$ である。したがって、高速コーディングのための完全なアルゴリズムが、付録3に示されている。

【0053】これらのアルゴリズムは、非常に簡易に実行することができる。特に、非常に少ないパラメータ、すなわち、数列 w 内の「1」の $(n-m+1)(t-1)$ 個の非ゼロ位置、および、あるいは $k2$ 個のコーディング係数によって、コードを定義する特徴を有している。それらのアルゴリズムは、条件 $a-d$ (例えば、長さ $P=25$ の $n=6$ 個の数列 w を必要とする、冗長度 $6/5$ の(150, 125)コードではないこと)に合致する可能なコードの全てを与えるものではないが、 N および K が事前に定義されている任意のアプリケーションにおいて、

— $NLDPC=N$, $KLDPC=K$ を有する($NLDPC$, $KLDPC$)コード、または、

— 任意にゼロに設定された d 個の有用なシンボルの非伝達により短縮される微小の d を有する($NLDPC+d$, $KLDPC+d$)近接コードのいずれかを発見することができる。

【0054】例えば、冗長度 $5/3$ (率0.6)のコード(N, K)を得るためには、 $d=NLDPC/15$ を有する冗長度 $8/5$ (率0.625)の($NLDPC+d$, $KLDPC+d$)コードから開始するだけで十分である。500以下の N 値および $t=3$ に対して、以下の冗長度を有する932個の異なるコードを極めて迅速に構築することが可能である(ここでは、故意に、 $4\sim 8/7$ の間に配される冗長度および $k2=3$ に対して $w[0]=\{1, 1, 0, 1, 0, 0, 0, \dots\}$ のコードに制限した。)

$R=20/17$ すなわち、1. 176 (2 コード)

$R=21/18$ または $7/6$ すなわち、1. 167 (17 コード)

$R=24/21$ または $8/7$ すなわち、1. 143 (3 コード)

【0055】さらに、500以下のNの所定値に対して、(N=480に対して)12個までの異なるコードが存在する。例えば、Nが288以上の6の倍数になるとすぐに、長さN、冗長度 $6/5$ 、 $3/2$ 、 $2/1$ の3つのコード、例えば、LDPC(300, 250)+LDPC(300, 200)+LDPC(300, 150)が常に存在する。

【0056】このことは、それぞれが長さNを有し、異なる感度を有する3つのバイナリ数列から構成されたバ

イナリ数列を効果的に保護するのに非常に有用である。もちろん、例えば、マトリクスAの行および/または列のランダム順列のような、これらのアルゴリズムの種々の変形例を想定することも常に可能である。また、非バイナリコードへの適合が特に簡易であることも重要である。

【0057】

【付録】

付録 1

```
for(x=0; x<n-m+1; x++){
  pos[x][0] = 0;
  for(pos[x][1]=pos[x][0]+1; pos[x][1]<M-1;
    pos[x][1]++){
    for(pos[x][2]=pos[x][1]+1; pos[x][2]<M;
      pos[x][2]++){
      (if the conditions are not satisfied,
        continue
        otherwise, go to ok)
    }
  }
  (stop: impossible to find a suitable choice
  for pos[x][0...t-1])
ok;;
}
```

付録2

(language C: the operator "^" corresponds to EXCLUSIVE OR.

```

/* Initialization of the b pass, of length M*/
for(i=M-k2; i<M; i++)
    b[i]=0;

/* Calculation of N successive values of the
impulse response of 1/A(z)*/
b[0] = 1;
for(i=1; i<k2; i++)
    b[i] = b[(i+M-(k2-k1)) % M]^b[(i+M-k2) % M];
for(i=k2; i<M; i++)
    b[i] = b[i-(k2-k1)]^b[i-k2];

/* Arrange for there to be just one 1
in the last k2 positions of b filtered by A(z)*/

weight = 0; /* all except 1 */

while(weight != 1){
    /* Shift by one notch */
    for(i=1; i<M; i++)
        b[i-1] = b[i];
    b[M-1] = b[M-1-(k2-k1)]^b[M-1-k2];
    /* Verify */
    weight = 0;
    for(i=M-k2; i<M; i++){
        char sum = b[i]^b[(i+k1)%M]^b[(i+k2)%M];
        if(sum){
            shift = M - i;
            weight++;
        }
    }
}

/* Particular case where M is forbidden */
if(weight ==0)
    return(M_FORBIDDEN);

```

```

}

/* rightward final circular shift:
b[i]=b[(i - shift) % M]*/
for(dec=0; dec<shift; dec++){
    char temp = b[M-1];
    for(i=M-1; i>0; i--){
        b[i]=b[i - 1];
    }
    b[0] = temp;
}

return(OK);

```

付録 3

(language C):

/* Phase 1: calculation of M intermediate parities z.

These parities are calculated by reading the successive columns of the coding matrix, namely A[*] [M], ..., A[*] [N]

They are placed at the head of x temporarily */

```

#define      z      x
    for(i=0; i<M; i++)
        z[i] = 0;

```

```

/* Loop over the n-m right submatrices of A*/
c0 = M;
c1 = c0 + P;

```

```

for(k = 1; k<=n - m; k++){
    offset = 0;
    for(c=c0; c<c1; c++){
        if(x[c]!=0)
            for(i=0; i<t; i++){
                /*      p      ought      to      be
                   offset + pos[i].
                   We decrement it by k2 to avoid
shifting
                   the array z before phase 3*/
                p = offset + pos[k*t+i] - k2;
                if(p<0)
                    z[p + M] = z[p + M]^1;
                else
                    if(p<M)
                        z[p] = z[p]^1;
                    else
                        z[p - M]=z[p - M]^1;
            }
        offset = offset + m;
    }
}

```

```

    c0 = c1;
    c1 = c1 + P;
}

/* Phase 2: calculation of the last k2 parity
symbols */
ixb0 = M - 1 - k2;

/*1: initialization of the last k2 elements of y
temp[0...k2-1]=y[M-1, M-2, ...M-k2]*/
for(k=0; k<k2; k++)
    temp[k] = 0;

/*2: copy over the last k2 elements of b
reg[0...k2-1] = b[M-k2...M-1]*/
for(i=0; i<k2; i++)
    reg[i] = finB[i];

/*3: iterative calculation of the last k2 symbols
*/
for(i=0; i<M; i++){
    /* b[i] = {1 0 0 ...}^b[i-(k2-k1)]^b[i-k2]
    with b[i-k2]...b[i-1] = reg[0...t2-1]

```

We must verify that:

```

b[-k2] + b[k1-k2] + b[0] = 0
...
b[-2] + b[k1-2] + b[k2-2] = 0
b[-1] + b[k1-1] + b[k2-1] = 0
b[0] + b[k1] + b[k2] = 1
b[1] + b[1+k1] + b[1+k2] = 0
... */
if(i==k2)
    input = 1;
else
    input = 0;
bi = input^reg[0]^reg[k1];
for(k=1; k<k2; k++)
    reg[k - 1] = reg[k];

```

```

reg[k2 - 1] = bi;

if(bi != 0)
    for(k=0; k<k2; k++)
        if(z[(ixb0 - k + M) % M] != 0)
            temp[k] = temp[k]^1;
ixb0 = ixb0 + 1;
if(ixb0==M)
    ixb0 = 0;
}

/*4: The z values have already been left-shifted
to avoid overwriting. Otherwise, it would be
necessary to do:

for(k=0; k<M - k2; k++)
    z[k] = z[k + k2];

Copy over temp to the end of y */
#define y      x
for(k=0; k<k2; k++)
    y[M - 1 - k] = temp[k];

/* Phase 3: calculation of y[M-k2-1, M-k2-2, ..., 0]
y[k+k2-k2] + y[k+k2-k1] + y[k+k2-0] + z[k+k2] = 0
y[k] goes to x[k]
z[k+k2] is in x[k]
Hence:
x[k+k2-k2] + x[k+k2-k1] + x[k+k2-0] + x[k] = 0
i.e.:
x[k+k2-k2] = -(x[k+k2-k1] + x[k+k2] + x[k])
*/
for(k = M-k2-1; k>=0; k--)
    y[k] = y[k+k2-k1]^y[k+k2]^z[k];

```

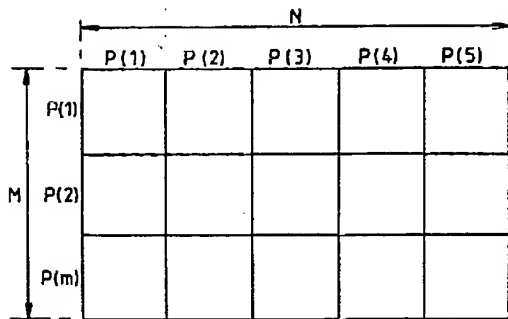
【図面の簡単な説明】

- 【図1】 検査マトリクスAの分割を示す配列である。
【図2】 図1の配列の m^2 左サブマトリクスの $M \times M$ サブマトリクスへのグループ化を示す図である。
【図3】 図1の配列の m^2 左サブマトリクスの $M \times P$ サブマトリクスへのグループ化を示す図である。
【図4】 この発明に係るプロセスの第1の変形例に従う検査マトリクスAを示す図である。
【図5】 この発明に係るプロセスの第2の変形例に従

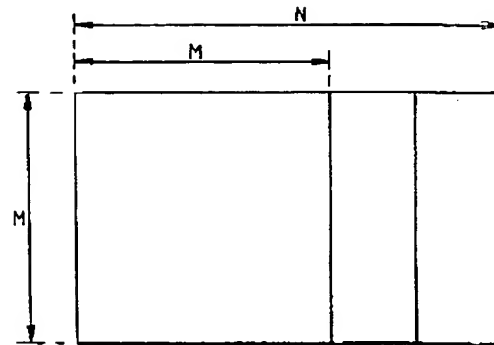
う検査マトリクスAを示す図である。

- 【図6】 冗長シンボルを計算するためのスイッチングマトリクスを示す図である。
【図7】 図6と同様のスイッチングマトリクスを示す図である。
【図8】 図6と同様のスイッチングマトリクスを示す図である。
【図9】 図6と同様のスイッチングマトリクスを示す図である。

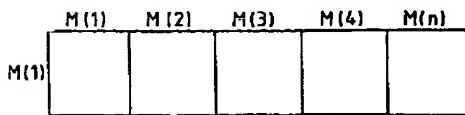
【圖1】



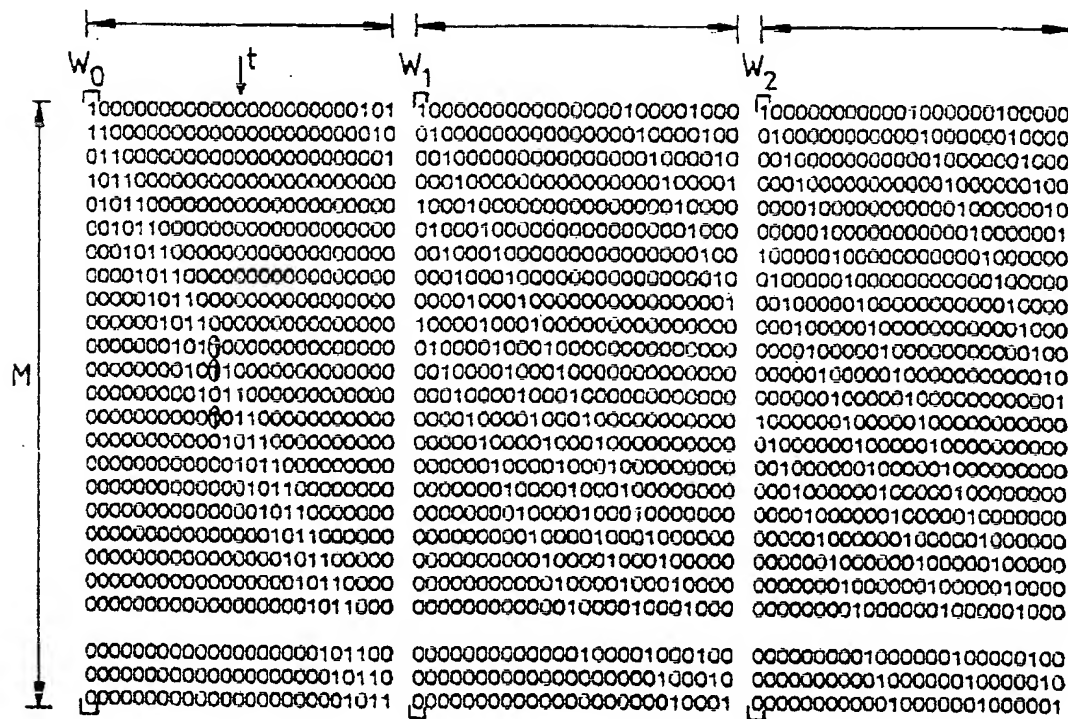
【圖2】



【圖3】



【圖4】



【図5】

[illegible]

【図6】

[illegible]

【図7】

Y[0]	Y[M-1]	Z[0]	Z[M-1]
10000000000000000000101		10000000000000000000000	
110000000000000000000010		01000000000000000000000	
011000000000000000000001		00100000000000000000000	
101100000000000000000000		00010000000000000000000	
010110000000000000000000		00001000000000000000000	
001011000000000000000000		00000100000000000000000	
000101100000000000000000		00000010000000000000000	
000010110000000000000000		00000001000000000000000	
000001011000000000000000		00000000100000000000000	
000000101100000000000000		00000000010000000000000	
000000010110000000000000		00000000001000000000000	
000000001011000000000000		00000000000100000000000	
000000000101100000000000		00000000000010000000000	
000000000010110000000000		00000000000001000000000	
000000000001011000000000		00000000000000100000000	
000000000000101100000000		00000000000000010000000	
000000000000010110000000		00000000000000001000000	
000000000000001011000000		00000000000000000100000	
000000000000000101100000		00000000000000000010000	
000000000000000010110000		00000000000000000001000	
000000000000000001011000		00000000000000000000100	
000000000000000000101100		00000000000000000000010	
000000000000000000010110		00000000000000000000001	
000000000000000000001011			
00000000000000000000101100		000000000000000000001000	
00000000000000000000101100		0000000000000000000000100	
0000000000000000000010110		0000000000000000000000010	
000000000000000000001011		0000000000000000000000001	

【図8】

Y[0]	Y[M-1]	Z[0]	Z[M-1]
10000000000000000000000		1101110010111001011100101	
01000000000000000000000		1110111001011100101110010	
00100000000000000000000		0111011100101110010111001	
00010000000000000000000		1011101110010111001011100	
00001000000000000000000		0101110111001011100101110	
00000100000000000000000		0010111011100101110010111	
00000010000000000000000		1001011101110010111001011	
00000001000000000000000		1100101110111001011100101	
00000000100000000000000		1110010111011100101110010	
00000000010000000000000		0111001011101110010111001	
00000000001000000000000		1011100101110111001011100	
00000000000100000000000		0101110010111011100101110	
00000000000010000000000		0010111001011101110010111	
00000000000001000000000		1001011100101110111001011	
00000000000000100000000		1100101110010111011100101	
00000000000000010000000		1110010111001011101110010	
00000000000000001000000		0111001011100101110111001	
00000000000000000100000		1011100101110010111011100	
00000000000000000010000		0101110010111001011101110	
00000000000000000001000		0010111001011100101110111	
00000000000000000000100		1001011100101110010111011	
00000000000000000000010		1100101110010111001011101	
00000000000000000000001		1110010111001011100101110	
000000000000000000000001		0111001011100101110010111	
000000000000000000000000		1011100101110010111001011	

【図9】

Y[0]	Y[M-1]	Z[0]	Z[M-1]
101100000000000000000000		000100000000000000000000	
010110000000000000000000		000010000000000000000000	
001011000000000000000000		000001000000000000000000	
000101100000000000000000		000000100000000000000000	
000010110000000000000000		000000010000000000000000	
000001011000000000000000		000000001000000000000000	
000000101100000000000000		000000000100000000000000	
000000010110000000000000		000000000010000000000000	
000000001011000000000000		000000000001000000000000	
000000000101100000000000		000000000000100000000000	
000000000010110000000000		000000000000010000000000	
000000000001011000000000		000000000000001000000000	
000000000000101100000000		000000000000000100000000	
000000000000010110000000		000000000000000010000000	
000000000000001011000000		000000000000000001000000	
000000000000000101100000		000000000000000000100000	
000000000000000010110000		000000000000000000010000	
000000000000000001011000		000000000000000000001000	
000000000000000000101100		000000000000000000000100	
000000000000000000010110		000000000000000000000010	
000000000000000000001011		000000000000000000000001	
000000000000000000000100		1110010110010110010110	
000000000000000000000010		011100101110010111001011	
000000000000000000000001		1011100101110010111001011	